Šolski center Celje
Splošna in strokovna gimnazija Lava

Risanje grafov

(Raziskovalna naloga)

Mentor:
Mojmir KLOVAR, univ. dipl. inž.

Avtorja:
Georg HALUŽAN, GL - 4. F
Andrej HERCOG, GL - 4. F

Celje, marec 2006

Povzetek

Raziskovalna naloga obsega opis programa za risanje grafov. Program je napisan v programskem jeziku C++. V programski kodi je uporabljen algoritem za pretvorbo niza v postfiksni zapis, s pomočjo katerega program računa izraze. Program deluje tako, da lahko uporabnik vpiše željeno fukcijo, v prostor, ki je namenjen za to, in program mu izriše graf funkcije. Pri tem obstaja možnost izbire povečave glede na kordinatno izhodišče ter barvo grafa.

Kazalo

Povzetek 2
Kazalo 3
1. Uvod 4
1.1. Predstavitev raziskovalnega problema 4
1.2. Hipoteza in cilji 4
1.3. Raziskovalne metode 4
2. Opis izdelave programa 5
3. Postfiksna metoda 6
4. Razlaga algoritma 7
5. Delovanje sklada 10
6. Preverjanje pravilnosti izraza 11
7. Izris funkcije 11
8. Dodatki k programu 12
8.1. Povečava 12
8.2. Barva grafa 13
8.3. Brisanje grafov 13
9. Poprava izrisa 13
10. Zaključek 14
11. Zahvala 15
12. Viri 16

Kazalo slik

Slika 1: Program 5
Slika 2: Primer sklada (3 * 4 = 12) 10
Slika 3: Primer sklada (5 * 4 + 3 = 23) 11
Slika 4: Spreminjanje povečave grafa 12
Slika 5: Spreminjanje barve grafa 13
Slika 6: Izvršilni gumbi 13

Kazalo tabel

Tabela 1: Prioriteta operatorjev 6

1. Uvod

1.1. Predstavitev raziskovalnega problema

Za raziskovalno nalogo sva izdelala program za risanje grafov. Program deluje tako, da sam prepozna enačbo, jo preveri in nariše. Program zajema polinomske in racionalne enačbe.

1.2. Hipoteza in cilji

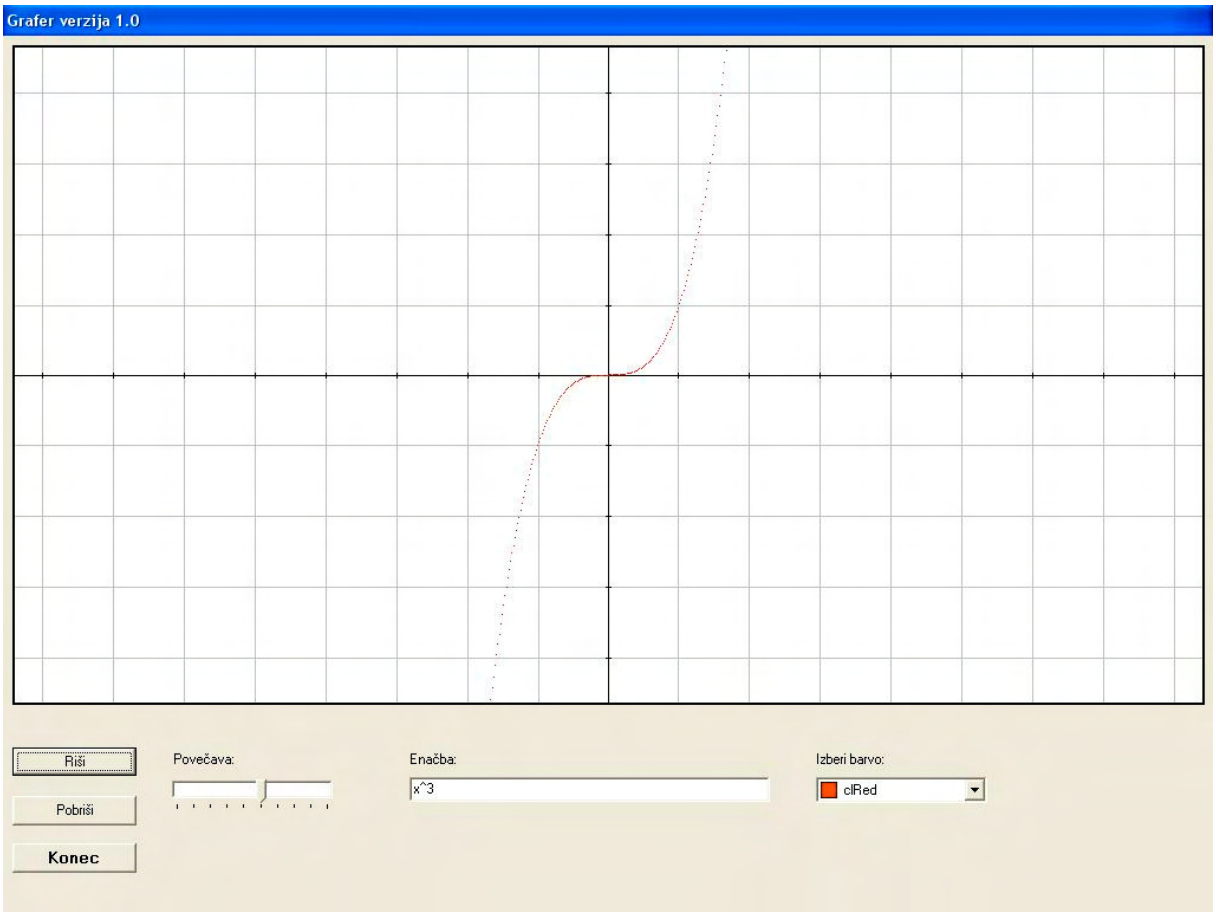
Po pogovoru z mentorjem sva se odločila, da bova naredila program, ki prebere enačbo, ki jo uporabnik vpiše kot tekst, in preveri, če je zapis pravilen. Če enačba ustreza pogojem, zapis pretvori v polski postfiksni zapis, ga izračuna ter nariše. Pri tem lahko uporabnik določi velikost enote na kordinatnem sistemu in razdaljo med točkama, ki jih računalnik izračuna za izris.

1.3. Raziskovalne metode

Raziskovanla naloga je izdelana v programu Borland C++ Builder 6.0. Ta program omogoča tudi vizualno okolje, v katerem je raziskovalna naloga napisana, saj se pri pouku računalništva učimo osnov tega programskega jezika.

2. Opis izdelave programa

Osnovo za raziskovalno nalogo sva dobila pri mentorju. Mentor je za demonstracijo izdelal program, za risanje funkcije, ki jo mora programer predhodno zapisati v sintakso programa. Najina predhodnika sta ta program nadgradila s tem, da je enačbo lahko vpisal uporabnik, vendar je moral predhodno označiti, katerega tipa je funkcija. Midva sva se odločila , da uporabiva program, ki ga je izdelal mentor, vendar ga nadgradiva tako, da uporabnik vpiše enačbo fukcije, ki jo program nato sam prepozna, izračuna in nariše. Pri tem sva morala uporabiti algoritem, ki enačbo, v znakovnem nizu pretvori v posfiksni zapis. Nato mu podava vrednosti za neodvisno spremenljivko. Program računa vrednost vpisane spremenljivke, in s tem dobi program koordinate, za izris grafa.



Slika 1: Program

3. Postfiksna metoda

Če računamo z žepnim kalkulatorjem, vanj vpišemo običajen izraz, v katerem je operator med dvema operandoma.

(a + b) * c + d + f * h

Pri posftiksnem zapisu pa morajo operatorji stati za operandi.

(a b +) c * d f h * + +

Program sva morala narediti tako, da zapis, v katerem so operatorji med operandi, pretvorit v postfiksni zapis. Pomagala sva si s funkcijo, v kateri sva morala paziti na prioriteto operatorjev, kajti * ima večjo prioriteto kot +. To pa pomeni, da se izračun operandov, v katerem nastopa *, izvrši pred izvršitvijo ukaza, v katerem nastopa +. Ker imamo v matematiki več različnih operatorjev z različnimi prioritetami, sva si naredila tabelo, v kateri imam zapisano prioriteto operatojev.

operatorji	prednosti
+, -	1
*, /	2
^	3
(	4
)	5

Tabela 1: Prioriteta operatorjev

Prednost pomeni število, ki se shrani na določeno spremenljivko, za preverjanje prednosti pri postavitvi operatorjev v niz, ki ga uporabnik vpiše.

```
int u_prioriteta(stack **k1, char x)
        {int na_skladu = 0, na_vrsti = 0;
         stack *d;
         d = *k1;
         if (d != NULL)
              {switch (d->x)
                    {case '+':
                     case '-': na_skladu = 1; break;
                     case '*':
                     case '/': na_skladu = 2; break;
                     case '^': na_skladu = 3; break;
```

```
                case '(': na_skladu = 4; break;
                case ')': na_skladu = 5; break;
               }
          switch (x)
               {case '+':
                case '-': na_vrsti = 1; break;
                case '*':
                case '/': na_vrsti = 2; break;
                case '^': na_vrsti = 3; break;
                case '(': na_vrsti = 4; break;
                case ')': na_vrsti = 5; break;
               }
         }
     else return 0;
     if(na_vrsti <= na_skladu)return 1;
     else return 0;
    }
```

4. Razlaga algoritma

Pri algoritmu sva za zapisovanje operandov in operatorjev potrebovala seznam in sklad. Seznam sva poimenovala izhod, nanj sva shranjevala operande in operatorje, ki so imeli enako ali večjo prioriteto kot naslednji operator v nizu. Na sklad pa sva shranjevala operatorje.

Pretvarjanje iz tekstovnega zapisa v polski posfiksni zapis sva naredila v zanki s pomočjo stavka switch. Funkcija pregleduje vsak znak v nizu od začetka do konca. Ko je funkcija prišla do operanda, ga zapiše na izhod, če pa je prišla do operatorja, ga zapiše na sklad. Pri tem sva morala paziti na prioriteto operatorjev, kajti če sta prišla dva operatorja iste prioritete na sklad, sva morala prejšnji operator vpisati na izhod in ga na skladu zbrisati.

```
Prebere "2"
 Doda "2" na izhod
  Izhod: 3 4 2
  Sklad: + *

Prebere "/"
 Pobriše "*" na sklad in ga doda na izhod, doda "/" na sklad
```

```
  Izhod: 3 4 2 *
  Sklad: + /
```

Ko je zanka prišla do znaka NULL, je bilo treba vse operatorje na skladu prepisati v obratnem vrstnem redu na izhod.

```
Prebere "2"
 Doda "2" na izhod
  Izhod: 3 4 2 * 1 5 - 2
  Sklad: + / ^

Konec algoritma
 Prestavi vse operatorje, v obratnem vrstnem redu iz sklad na izhod in
sklad izbriše
 Izhod: 3 4 2 * 1 5 - 2 ^ / +
```

Tukaj je primer tega uporabljenega algoritma. Niz predstavlja znakovni niz, ki ga uporabnik vpiše v ustezno tabelo.

```
Niz 3+4*2/(1-5)^2

prebere "3"
 Doda "3" na izhod
  Izhod: 3

prebere "+"
 Doda "+" na sklad
  Izhod: 3
  Sklad: +

Prebere "4"
 Doda "4" na izhod
  Izhod: 3 4
  Sklad: +

Prebere "*"
 Doda "*" na sklad
  Izhod: 3 4
  Sklad: + *
```

```
Prebere "2"
 Doda "2" na izhod
  Izhod: 3 4 2
  Sklad: + *

Prebere "/"
 Pobriše "*" na sklad in ga doda na izhod, doda "/" na sklad
  Izhod: 3 4 2 *
  Sklad: + /

Prebere "("
 Doda "(" na sklad
  Izhod: 3 4 2 *
  Sklad: + / (

Prebere "1"
 Doda "1" na izhod
  Izhod: 3 4 2 * 1
  Sklad: + / (

Prebere "-"
 Doda "-" na sklad
  Izhod: 3 4 2 * 1
  Sklad: + / ( -

Prebere "5"
 Doda "5" na izhod
  Izhod: 3 4 2 * 1 5
  Sklad: + / ( -

Prebere ")"
 Doda "-" na izhod in "-" ter "(" izbriše iz sklada. ")" ne zapiše na
sklad.
  Izhod: 3 4 2 * 1 5 -
  Sklad: + /

Prebere "^"
 Doda "^" na sklad
  Izhod: 3 4 2 * 1 5 -
  Sklad: + / ^
```

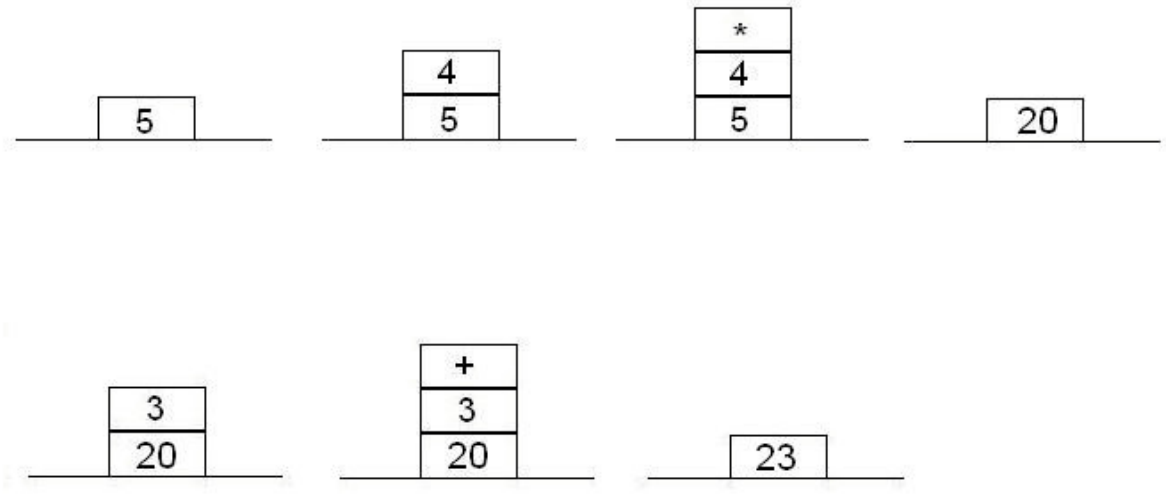
```
Prebere "2"
 Doda "2" na izhod
  Izhod: 3 4 2 * 1 5 - 2
  Sklad: + / ^

Konec algoritma
 Prestavi vse operatorje, v obratnem vrstnem redu iz sklada na izhod in
sklad izbiše
 Izhod: 3 4 2 * 1 5 - 2 ^ / +
```

5. Delovanje sklada

Pri algoritmu za pretvorbo sva uporabila sklad. Sklad je struktura, v kateri se operacije izvršujejo od zadnje, ki je bila vnešena , do prve. To si lahko predstavljamo kot zaboje, ki se zlagajo drug na drugega. Sklad sva potrebovala za izračun posfiksnega zapisa.

Ko program izdela posfiksni zapis, znak za znakom od leve proti desni, vpisuje na sklad. Pri tem se izračun izvrši, ko pride prvi operator na sklad. Na primer da je uporabnik vpisal 3 * 4, je to v posfiksni metodi 3 4 *. Torej se najprej na sklad vpiše 3, nato 4 in na koncu *. Ko je program zaznal operator na skladu, je operacijo izvedel, v tem primeru zmnožil 3 in 4. Vrednost tega izračuna ostane na skladu. Če bi imeli obsežnejši izraz, je rezultat tega izraza vrednost, ki ostane na skladu po zadnjem izvršenem ukazu.

Slika 2: Primer sklada (3 * 4 = 12)



Slika 3: Primer sklada (5 * 4 + 3 = 23)

6. Preverjanje pravilnosti izraza

Pri tem sva upoštevala matematično pravilnost izraza. Pozorna sva bila na predznak operandov, presledke med števili (npr. 3 4 namesto 34), oklepaje in decimalno vejico. Program je narejen tako, da če pride do napake pri vnosu enačbe, program javi, do katere napake je prišlo. Nato ima uporabnik možnost spremeniti zapis enačbe in narisati graf.

7. Izris funkcije

Za izris sva uporabila program, ki nama ga je predstavil mentor pred izbiro naslova za raziskovalno nalogo. Risanje opraviva s funkcijami canvasa. Najprej izriševa mrežo, nato koordinatni osi, enote na oseh in na koncu okvir, ki loči risalno polje od ozadja programa. Program je tako pripravljen za uporabo funkcije plot, s pomočjo katere izrisuje točke na zaslon. Funkciji plot je treba podati x in y koordinati, s pomočjo teh pa izriše točko.

```
void TForm1::Plot(float x, float y, TColor barva)
{
   float Xmax = Form1->ClientWidth;
   float Ymax = Form1->ClientHeight-100;
```

```
   float XC = Xmax/2;
   float YC = Ymax/2 + 100;
   float X = XC + x;
   float Y = YC - y;

   if ( ((X <= Form1->ClientWidth - 5)&&(Y <= ClientHeight-5)) && ((X >
5)&&(Y > 100)) )
      Canvas->Pixels[X][Y] = barva;
}
```

8. Dodatki k programu

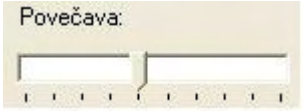
Program ima tri dodatke za boljšo preglednost izrisa:

- povečava
- barva grafa
- brisanje do sedaj narisanih grafov

Poleg teh dodatkov lahko uporabnik na isti koordinatni sistem nariše več grafov, kar pripomore k lažji primerjavi med grafi. Tudi samemu koordinatnemu sistemu sva dodala podaljške enot na abcisni in ordinatni osi. To nam omogoča, da ugotovimo, kje stojijo točke grafa, ki so oddaljene od središča koordinatnega sistema. Te črte so narisane s sivim odtenkom, da jih ločimo od abcisne in ordinatne osi.

8.1. Povečava

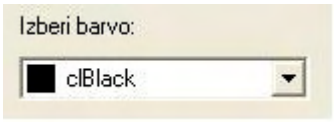
To je dodatek, pri katerem lahko uporabnik približa pogled koordinatnemu izhodišču s pomočjo drsnika. Tukaj mora program za vsako funkcijo, ki jo izriše, izrisati koordinatni sistem. In ker se koordinatni sistem spremeni, morava ponovno še izrisati graf funkcije.



Slika 4: Spreminjanje povečave grafa

8.2. Barva grafa

Program omogoča, da uporabnik določi barvo grafa, ki ga želi narisati. Za ta dodatek sva se odločila, saj ima program možnost risanja več grafov na isti koordinatni sistem, grafi pa se med seboj bolj ločijo, če so različne barve.



Slika 5: Spreminjanje barve grafa

8.3. Brisanje grafov

Tukaj lahko uporabnik pobriše vse grafe, ki so bili narisani na koordinatni sistem. Ni možno brisanje zadnjega narisanega grafa, ampak program pobriše vse do sedaj narisane. Poleg gumba za brisanje sta v programu še dva. Prvi gumb se imenuje »Riši«, s klikom miške nanj nariše graf funkcije, ki jo je uporabnik vpisal. Poleg teh dveh je še gumb »Konec«, s katerim zapustimo program.

Slika 6: Izvršilni gumbi

9. Poprava izrisa

Ker na canvas izrisujema točke, se pri velikih skokih med neodvisno spremenljivko pojavi prazen prostor. Da ta prazem prostor zapolniva in povežava narisane pike med sabo, uporabljava funkciji LineTo, MoveTo. Tako je najin graf videti kot povezana krivulja.

10. Zaključek

Ob začetku izdelave programa sva si zadala cilj, da bi najin program uporabljal postfiksno metodo, za izračun izrazov in da bi program znal narisati vse polinome. To nama je tudi uspelo in sva s pomočjo mentorja še to nadgradila. Sedaj program riše linearno funkcijo, kvadratno fukncijo, racionalno funkcijo, polinome, eksponentno funkcijo in potenčno funkcijo. Odločila sva se da bova ta program še nadgradila, tako da bo poleg vseh teh funkcij znal narisati še logaritemsko funkcijo, korensko funkcijo in vse kotne funkcije.

11. Zahvala

Zahvaljujeva se najinemu mentorju za pomoč pri programiranju in usmerjanju pri izdelavi raziskovalne naloge. Zahvala gre tudi vsem ostalim, ki so na kakršenkoli način pripomogli, k boljši raziskovalni nalogi.

12. Viri

Reverse Polish notation

Pridobljeno 4.2.2006 iz

http://en.wikipedia.org/wiki/Reverse_Polish_notation.

The reverse Polish algorthm for hewlett packard calculators

Prodobljeno 4.2.2006 iz

http://users.ece.gatech.edu/~mleach/revpol/.

Algoritmi in podatkovne strukture

Pridobljenio 16.2.2006 iz

http://rts.uni-mb.si/misc/aps/index.html.

Kozak, J. (1986). Podatkovne strukture in algoritmi. Ljubljana: Društvo matematikov, fizikov in astronomov SRS.